Source Code Partitioning Using Process Mining

Koki Kato[1], Tsuyoshi Kanai[1], and Sanya Uehara[2]

[1] Software Innovation Lab., Fujitsu Laboratories Ltd., Japan
[2] Fujitsu Ltd., Japan
{kato_koki,kanai.go,s.uehara}@jp.fujitsu.com

Abstract. Software maintenance of business application software such as adding new functions and anti-aging should be performed cost-effectively. Information such as grouping of business activities that are executed as a unit, source code which corresponds to the activities, and the execution volume of the activities is useful for deciding on what areas of business application software to invest in, and prioritizing maintenance requests.

We propose a new method which extracts such information using the BPM-E process mining tool we have developed.

The method was applied to in-house business systems; the results showed that the method successfully extracted the grouping of events, but that there are accuracy issues in associating events with source code.

Keywords: process mining, source code analysis, business application maintenance.

1 Introduction

Business application software in enterprises requires new features and functions to adapt to changing customers' needs and corporate strategies. Adaptive maintenance[2] is applied to business applications to add new business functions for adapting to business trend. Adaptive maintenance requires more human resources than developing the application itself when the application is used for a long time[8]. As there may be many modification requests to fulfill, it is necessary to prioritize them within budget. It is also important to estimate the extent of the impact to source code when adding a new business function.

Maintenance of aging software tends to become more difficult (and expensive), since updates gradually destroy the original structure of the applications[8]. Anti-aging methods such as refactoring and restructuring are kinds of perfective maintenance[2] which improve the maintainability[5]. If perfective maintenance is applied, cost reduction for function addition is expected, though cost for perfective maintenance itself is needed. To perform anti-aging methods cost-effectively, they should be applied only to those parts of the source code that are related to a group of business activities that are executed as a unit, and many changes to the units are expected in the future. We call such a unit of business activities a 'work package' in this paper. It can be said that a work package that the execution volume is in the increasing tendency is growing up on business,

S. Rinderle-Ma, F. Toumani, and K. Wolf (Eds.): BPM 2011, LNCS 6896, pp. 38–49, 2011.

and it is expected that new functions are added to such a work package. Therefore, a new technology is necessary for extracting work packages, their execution volume, and relating work packages to source code, to examine the priorities of adaptive and perfective maintenance in terms of work packages, based on the tendency to the business execution.

Process mining[3] is a technology for analyzing business operations based on event logs, and can extract useful data such as the types of events and their execution volume without placing any load on the business system.

In this paper, we propose a new method which extracts work packages using BPM-E, which is a process mining tool, and associates work packages with portions of source code of business application(s). After introducing BPM-E, a method for extracting work packages and a method for extracting source code corresponding to the events are described. We show and discuss the results of applying the method to two in-house business systems.

2 BPM-E

Business Process Management by Evidence (BPM-E, or Automated Process Discovery (APD))[1] is a process mining tool that has been developed at Fujitsu Laboratories since 2004, and has already been used to analyze customers' business in the US, Europe and Japan.

One of the distinguishing features of BPM-E is that it extracts business process flows directly from copies of transactional data (in Comma Separated Value (CSV) format); there is no need to convert data to a specific input format, and it is easy to dump databases in CSV format.

2.1 Events

A business activity is called an *event* in BPM-E. An event is expressed by three kinds of information $(id, type, timestamp)$, where id is a business identifier (such as 'ORD1234567' of an 'order number'), $type$ is an identifier that shows which kind of event was generated (such as 'acceptance of order' and 'packaging') and is also called an 'event class', and $timestamp$ is the time at which the event occurred.

2.2 Classification of Tables

There are two main types of table in enterprise systems:

- Master data tables: These maintain lists of identifiers such as bills of materials (BOM), and are updated infrequently.
- Transaction tables: These contain data (such as IDs and time-stamps) of daily business operations.

BPM-E uses transaction tables as input data.

The transaction tables are categorized as follows based on the table structure:

- Log/history type: An event is recorded as a record which contains at least an ID, a time-stamp, and a type of event (an event class). There may be multiple records for one ID in a table. The event class for a record may be distinguished by the combination of multiple column values.
- No column for storing event class values: There are two ways to express the event class:
  - The table name indicates the event class: Events of a certain event class are recorded in a the dedicated table (e.g. a table for 'acceptance of orders'). The event class is identified by the table name.
  - The column name indicates the event class: A table has multiple time-stamp columns. When an event of a certain event class arises, the corresponding column of the record is updated with the time-stamp.
- Summary table for progress reporting: A table for progress reporting, which has multiple time-stamp columns. Each time-stamp column corresponds to an event class. This is a variation of the above 'the column name indicates the event class', which differs in that the time-stamp values are duplicated ones; the values are written together with other transaction tables, or the values are replicated from other tables periodically.

The above-mentioned types may exist together, and BPM-E can process them.

The main analysis functions of BPM-E include the business process flow diagram, the exceptional flow diagram, the narrow-down search, and the flow type display.

3 Problem and Notations

In this section, we define a problem and some notations. We assume that the business applications are written in Java and SQL.

When using process mining tools such as BPM-E, we sometimes observe that some events (activities) form a unit, and such a unit is executed in different flow types. We call such a unit a 'work package' in this paper. A work package G_i is defined as a set of event classes $\{E_{i_1}, E_{i_2}, \ldots\}$, and an event class belongs to only one work package.

As events of a work package have strong ties from the viewpoint of business execution, it is presumed that source code that relates to events of a work package also has strong relation. Therefore, the influence is expected to occur by the unit of work package when adding a function or doing refactoring. Source code that relates to a work package may not correspond to the static source structure; sometimes a work package and corresponding source code may extend over multiple business systems.

When doing maintenance such as adding functions and refactoring only to the source code corresponding to the work package, cost reduction is expected because source code which should be surveyed can be limited. On the other hand, there is source code like utility which is called from various points; when such source code is changed, the change influences other source code.

Therefore, we would like to solve a problem of deriving a set of work packages, source code (methods) that corresponds to work package, and a set of commonly used methods corresponding to common events from flow types (as well as their execution volume) and a set of source code. The problem can be described as follows:

Problem 1. Find a tuple (G, M_G, CM) from the tuple $(F, code)$, where,
$G = \{G_0, G_1, \ldots\}$ is the set of work packages,
$M_G = \{M_{G_0}, M_{G_1}, \ldots\}$ is the set of sets of Java methods corresponding to work packages,
CM is the set of common methods (described later),
$F = \{F_1, F_2, \ldots, F_M\}, \#(F_i) \geq \#(F_j)$ for $i > j$ is the set of business process flow types, which has M elements, and $\#(F_i)$ is the number of occurrences of business process flow type F_i,
code is the set of source code of the business application(s),
M_{G_i} is a set of methods corresponding to the work package G_i, and
$F_i = (E_{i_1}, E_{i_2}, \ldots, E_{i_{n_i}})$ is a business process flow type.

4 Processing Procedures

We propose the following procedure for solving the problem:

1. Extract work packages from business process flow types.
2. Estimate the source code which corresponds to events directly.
3. Determine the relevant source code among the above source code.

4.1 Extracting Work Packages

When we examine business process flow types, we find that there are two types of work packages. One is a work package that characterizes flow type(s); for instance in order-receiving system, manufacturing-request operation may characterize the business operation at stock shortage. Another is a work package that does not depend on the variation of the business; for instance, accepting-order operation will almost always occur in spite of stock condition or method of delivery. We classify work packages as follows:

- Common events (G_0): The work package which is executed in almost all business process flow types. Common events may not exist according to circumstances.
- (Other) work packages ($G_1, G_2, \ldots$): the sets of event classes which are specific to business operations.

The work packages are extracted as follows:

1. Extract common events.
2. For the remaining events, create groups of the events using the business process flow types in order of the number of executions.
3. Compensate the groups.

```
P ← G_0
for i = 1 to M do
  G_i ← {}
  for j = 1 to n_i do
    if E_ij ∉ P then
      G_i ← G_i + E_ij
      P ← P + E_ij
    end if
  end for
end for
```

(a) Extracting work packages.

```
for j = M downto 2 do
  for i = 1 to M do
    C_i ← 0 {clear counter}
  end for
  for i = 1 to M do
    if F_i ⊃ G_j then
      for k = 1 to j − 1 do
        if F_i ⊃ G_k then
          C_k ← C_k + #(F_i)
        end if
      end for
    end if
  end for
  C_total ← Σ_{i=1}^{M} C_i
  C_max ← max(C_1, ..., C_M)
  maxPos ← x, where C_max = C_x
  if C_max/C_total > τ_m then
    Gmove.put(j, maxPos)
  end if
end for
for j = M downto 1 do
  maxPos ← Gmove.get(j)
  if maxPos ≠ null then
    G_maxPos ← G_maxPos + G_j
    G_j ← {}
  end if
end for
```

(b) Compensation.

Fig. 1. Algorithm of extracting work packages

Extraction of Common Events. The common events are calculated and set as G_0 by finding the event classes for which the ratio of the business process flow type including an event class is more than a threshold (τ_c).

Extraction of Work Packages. We assume that the larger the execution volume of a certain business process flow type, the more important for the enterprise. Therefore, when extracting the work packages it is appropriate to put priority on business process flow types with larger execution volume.

Then, a set of event classes which are not yet classified are registered as a work package in the order of execution frequency of the business process flow type. The pseudo code is shown in Fig. 1 (a). In the figure, P is a variable that stores already processed event classes.

Compensation Process. The work packages obtained in the above step may contain those that should be merged with other work packages to obtain larger work packages. Therefore, from the last registered work package, if the business process flow type contains a work package and another work package with high

frequency (i.e. the ratio is more than τ_m), these work packages are merged. The pseudo code is shown in Fig. 1 (b). In the figure, 'Gmove' is an associative memory; $Gmove.put(i, j)$ stores j in association with i, and $Gmove.get(i)$ obtains the value associated with i (i.e. j). C_k accumulates the number of occurrences of flow types which include all events of work package G_k as well as G_j for $k < j$. The ratio C_{max}/C_{total} means how well both G_{maxPos} and G_j are included in flow instances simultaneously, and whether G_{maxPos} is able to merge with G_j.

Execution volume for each work package is estimated by using execution volume of events.

4.2 Estimating the Source Code Which Corresponds to Events Directly

The estimation is done using the fact that the events of BPM-E, which correspond to the data of the table(s), are generated by the write operations (execution of `INSERT`/`UPDATE`), according to the SQL `INSERT`/`UPDATE` statements in Java methods. An event should be related to a method that generates `INSERT` or `UPDATE` statement, not a method that executes SQL statements, because the latter method may be the common utility invoked from different methods with parameters which have SQL statements.

Methods which correspond to events directly are extracted as follows:

1. Extract the tuple $(SQLstatement, m)$ from the source code, where *SQL statement* is a string that is grammatically correct and contains the string "`INSERT`" or "`UPDATE`", and m is the method which contains the above SQL statement.
2. Extract the table name, time-stamp column name, and event class which corresponds to the event class E_i from the BPM-E definition information.
3. When the above relation exists, store the method m in M_{G_i}. Note that a certain method may correspond to two or more event classes.

4.3 Determine the Relevant Source Code

The escalations are done for each method that is directly related to the event, by finding[1] caller methods and callee methods recursively as follows:

1. Do the following process with $M_{G_i}, i = 0, 1, \ldots$:
   (a) Find the methods that call the method m in M_{G_i} recursively. When an already registered method is reached, stop the search.
   (b) Add found methods to M_{G_i}.
2. Do the following process with $M_{G_i}, i = 0, 1, \ldots$,
   (a) Find the methods which are called from method m in M_{G_i} recursively, and store the newly found method m' and work package pair.

[1] We used Eclipse (http://www.eclipse.org/) AST (Abstract Syntax Tree).

3. If multiple packages are related to method m', then register method m' as the common method CM, otherwise add method m' to the corresponding work package $M_{G'_i}$.
4. Form a new work package and register all the methods that have not been registered yet.

5 Experimental Results

Two in-house business systems are used for the experiment. Both are written in Java and SQL without stored procedures. The features are shown in Table 1.

Table 1. System features

	System A	System B
Software architecture	Three-tiers + batch	Three-tiers
Framework	(not used)	Apache Struts + DbUtils
Number of classes	577	169
Number of methods	6872	3219
Lines of code	129744	31817
Number of event classes	36	7
Number of business process flow types	6810	11
Number of business process flow instances	307035	1709
Start year of development	2000	2008

5.1 Results of Work Package Extraction

The results are shown for system A only. We set the thresholds τ_c and τ_m to 0.8 by way of experiment.

The results of work package extraction are shown in Table 2 (displayed using aliases). Nine work packages are extracted including the common events.

Table 2. Results of work package extraction for System A

WP	Event classes
G_0	e1, e2, e3, e4, e5, e6
G_1	e7, e8, e9, e10, e11, e12, e13, e14, e15, e16, e17, e18
G_2	e19, e20, e21, e22, e23
G_3	e24, e25, e26, e27, e28, e29, e30, e31
G_4	e32

WP	Event classes
G_5	e33
G_6	e34
G_7	e35
G_8	e36

5.2 Results of Work Package–Source Code Relation

The numbers of methods which are associated with each work package for System A are shown in Table 3 (a). There are 27 duplicated methods in (a) because there were some methods related to multiple events. Similarly for System B, there are two duplicated methods in (b).

Table 3. Results of methods associated with work packages

(a) System A

Work packages	Number of methods
(Common Methods)	2600
G_0	3175
G_1	37
G_2	14
G_3	197
G_4	4
G_5	13
G_6	5
G_7	7
G_8	10
(Unrelated methods)	837

(b) System B

Work packages	Number of methods
(Common Methods)	584
G_0	0
G_1	98
G_2	1
G_3	3
(Unrelated methods)	2535

6 Discussion

First, we discuss the results of work package extraction compared with the business process flow diagrams drawn by the person in charge. Then, we discuss the accuracy of event–method relationships.

6.1 Evaluation of Work Package Extraction

To evaluate the presented method, we compared the results with manually extracted work packages from four business process flow diagrams (i.e. four flow types) which were drawn by the person in charge of System A.

Work Packages from Flow Diagrams for System A. One of the authors extracted the work packages from the diagrams manually based on the similarity of the activities (events) among the diagrams. The similarities between distantly-positioned activities within a diagram were not considered. The work packages and the business process flows expressed by work packages are shown in Tables 4 and 5, respectively. The right arrows indicate the execution order of the work packages.

To estimate the execution volume of the diagrams, the diagrams were compared with the business process flow type of BPM-E. The flow types were categorized using the activities in the diagrams and the flow types of the top 40 (the flow instance total is 218755). The results are shown in Table 5.

From Table 5, flow type A is the most frequently executed business process. Further analysis reveals two main types of variation of flow type A in 'not categorized' flows; one type contained $\Gamma 14$, and the other contained $\Gamma 12$. It is assumed that the person in charge did not describe the diagrams entirely.

Analysis of Work Packages. From Table 5, work packages that are common to flow types A–D are $\{\Gamma 1, \Gamma 4\} = \{e1, e4, e5, e6\}$.

Table 4. Manually extracted work packages and corresponding events for System A

WP	Corresponding events	WP	Corresponding events
$\Gamma 1$	$e4, e5, e6$	$\Gamma 8$	$e27$
$\Gamma 2$	$e14, e15$	$\Gamma 9$	$e34$
$\Gamma 3$	$e2, e7, e8, e9, e10, e12, e13, e17, e18$	$\Gamma 10$	$e3, e25, e30, e31, e35$
$\Gamma 4$	$e1$	$\Gamma 11$	$e24$
$\Gamma 5$	$e11$	$\Gamma 12$	$e19$
$\Gamma 6$	$e26, e29$	$\Gamma 13$	$e22$
$\Gamma 7$	$e20, e21, e23, e28, e32, e33, e36$	$\Gamma 14$	$e16$

Table 5. Flow type, work packages, and number of executions

Flow type	Flow using WP	Number of executions
Flow type A	$\Gamma 1 \rightarrow \Gamma 2 \rightarrow \Gamma 3 \rightarrow \Gamma 4 \rightarrow \Gamma 5$	79697
Flow type B	$\Gamma 1 \rightarrow \Gamma 6 \rightarrow \Gamma 7 \rightarrow \Gamma 8 \rightarrow \Gamma 9 \rightarrow \Gamma 10 \rightarrow \Gamma 2 \rightarrow \Gamma 3 \rightarrow \Gamma 4 \rightarrow \Gamma 5$	23564
Flow type C	$\Gamma 1 \rightarrow \Gamma 9 \rightarrow \Gamma 11 \rightarrow \Gamma 10 \rightarrow \Gamma 4 \rightarrow \Gamma 6 \rightarrow \Gamma 12 \rightarrow \Gamma 8$	11599
Flow type D	$\Gamma 13 \rightarrow \Gamma 1 \rightarrow \Gamma 7 \rightarrow \Gamma 3 \rightarrow \Gamma 4 \rightarrow \Gamma 12$	11522
(Not categorized)		92373

Compared with the above results, the common events (G_0) in Table 2 have extra events $\{e2, e3\}$. As for $e2$, as $e2$ is included in $\Gamma 3$ which is a part of flow type A and the non-categorized flow types, this result is considered to be valid.

As for $e3$, however, it is not common for the top 40 BPM-E flows, as there were only 13 flows which contained $e3$. However, there were 5541 flows which contained $e3$ in whole flows.

BPM-E treats the flows as different types of flows when the orders of the events differ, and/or the repeat times of an event differ (as the default). If there are business processes which produce variations, the above phenomenon may occur.

Table 6 shows the results of applying Table 4 to Table 2. From Table 6, we consider that the work packages G_0–G_3 represent the portion of business process flows that were accurately drawn by the person in charge, though it is hard to comment on G_4–G_8.

6.2 Evaluation of Event–Method Relationships

We evaluated the method from two viewpoints: the potential cover rate (the rate of methods that are related to events over total methods), and the accuracy of the event–method relationships.

Cover Rate. The ideal cover rate is 1, i.e. all methods are related to the events. A lower cover rate means that fewer methods can be estimated for the execution

Table 6. Work packages expressed by manually extracted work packages

G_0	$\Gamma 1 + \Gamma 4 + (e2) + (e3)$
G_1	$\Gamma 2 + (\Gamma 3 - e2) + \Gamma 5 + \Gamma 14 \approx$ a portion of type A and B.
G_2	$\Gamma 12 + \Gamma 13 + (e20 + e21 + e23) \approx$ a portion of type D.
G_3	$\Gamma 6 + \Gamma 8 + \Gamma 11 + (e25 + e28 + e30 + e31) \approx$ a portion of type C.
G_4	(no relevant matching)
G_5	(no relevant matching)
G_6	$\Gamma 9$
G_7	(no relevant matching)
G_8	(no relevant matching)

volume. The actual cover rate was 0.88 for System A and 0.21 for System B, showing a large difference between the systems.

According to the naming of the packages, the categories of methods that are not related are as follows:

- System A: batches, beans (which are not related to events), database, servlet, utilities, and CORBA. Concentration on specific packages or classes was not observed.
- System B: master maintenance, business logics, menus, and utilities.

The numbers of methods for each category on System B, classified into Apache Struts (Action and Form) and data processing, are as follows:

- Struts Action + Form [master maintenance: 89, business logic: 1648, menus: 520]
- data processing (including DB access) [master maintenance: 95, business logic: 159, utilities: 23]

There is some concentration on the methods corresponding to Struts Action and Form. Some ways to improve the cover rate for these might be:

- Analyzing the Struts configuration files to extract the relation between the methods for accessing web pages.
- Analyzing `SELECT` statements that read the data written by `INSERT`/`UPDATE`.

There are also some issues for extracting SQL statements. In both systems, there were SQL statements that are generated dynamically, such as building SQL statements using for-loops, and changing parameters using if-statements. To overcome these issues, we will need some techniques to execute partial Java code using an interpreter or virtual code execution, such as given in reference [4].

Evaluation of Events–Methods Relationships. Multiple events may be extracted for one method. The following two types cause such phenomenon.

- There are multiple `INSERT`/`UPDATE` statements in one method.
- One `INSERT`/`UPDATE` statement updates multiple time-stamp columns.

For the former case, the number of method executions is estimated as the total number of executions of events which are related to `INSERT/UPDATE`.

The latter case has issues, and System A includes this category. For a table structure type that has multiple time-stamp columns which correspond to events, the usual update processing may be that a record for a certain business ID is inserted first, and then a time-stamp column corresponding to an event is updated when an event occurs. However, especially for object-oriented languages like Java, a bean object may be used for storing a record data corresponding to a business ID, and all time-stamps are updated simultaneously when an event occurs.

This problem may be solved if we can pinpoint each method which sets the time-stamp to the bean, though some methods could not be specified on System A for the following reasons:

- The time-stamp value for each business activity was written into a temporary table each time an activity occurred, then the record in the temporary table was copied to the table that BPM-E used for event extraction.
- The implementation could not relate a method that set the time-stamp value into a bean and the SQL execution method when the bean was passed via Vector which contained multiple types of beans.

7 Related Work

One of the methods for estimating the execution volume and extracting portions of source code directly is to use trace logs or profiling information. In [9], a method called 'phase detection' is proposed to divide the trace logs, and the divided trace logs are digested into groups and visualized. The phases extracted by phase detection are similar to the work packages in this paper.

If we apply this kind of technology to real business systems, we will obtain phases and their number of executions accurately. However, it is not easy to acquire the trace logs (or to retrieve profile information) from a business system that is in operation for a long time, because doing so places a non-negligible load on the system. Meanwhile, the testing environment and test patterns for the business system development are not sufficient for obtaining trace logs or profiling, because the number of executions for each test pattern is quite different from the actual business environment, even though the test patterns are exhaustive.

Therefore, process mining such as BPM-E is practical for obtaining the statistics of business systems in operation rather than using trace logs, even though the approach is not as accurate as using trace logs.

Several studies [6] and [7] extracted groupings such as work packages using process mining. [6] proposed a method for describing events at a different level of abstraction, while [7] constructed a hierarchy of events clustering using the AHC algorithm.

8 Conclusions

We proposed a method for extracting work packages, their execution volume, and their relevant source code using process mining. The information can be used to develop a more cost-effective maintenance strategy for each work package.

The method was applied to two in-house business systems, and successfully extracted the common events and the work packages. However, there remain issues of accuracy in associating events with source code that arises from the extraction of dynamically generated SQL statements, and from acquiring the values which are set by beans.

To overcome such issues in a future study, we intend to investigate the virtual code execution (such as [4]) technology and apply it to code that generates SQL statements dynamically.

References

1. http://www.fujitsu.com/global/services/software/interstage/solutions/bpm/apd.html
2. International standard - iso/iec 14764 ieee std 14764-2006 software engineering — software life cycle processes — maintenance. ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998) (2006)
3. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., de Medeiros, A.K.A., Song, M., Verbeek, H.M.W.E.: Business process mining: An industrial application. Inf. Syst. 32(5), 713–732 (2007)
4. Brat, G., Havelund, K., Park, S., Visser, W.: Java pathfinder - second generation of a java model checker. In: Proc. of the Workshop on Advances in Verification (2000)
5. Carriere, J., Kazman, R., Ozkaya, I.: A cost-benefit framework for making architectural decisions in a business context. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010, vol. 2, pp. 149–157. ACM, New York (2010)
6. van Dongen, B.F., Adriansyah, A.: Process mining: Fuzzy clustering and performance visualization. In: BPM Workshops, pp. 158–169 (2009)
7. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: BPM Workshops, pp. 128–139 (2009)
8. Jones, C.: Geriatric issues of aging software. CrossTalk 20(12), 4–8 (2007)
9. Watanabe, Y., Ishio, T., Inoue, K.: Feature-level phase detection for execution trace using object cache. In: Proceedings of the 2008 International Workshop on Dynamic Analysis, WODA 2008, pp. 8–14. ACM, New York (2008)